PLP - Primer Parcial - 1er cuatrimestre de 2024

Este examen se aprueba obteniendo al menos dos ejercicios bien menos (B-) y uno regular (R). Las notas para cada ejercicio son: -, I, R, B-, B. Entregar cada ejercicio en hojas separadas. Poner nombre, apellido y número de orden en todas las hojas, y numerarlas. Se puede utilizar todo lo definido en las prácticas y todo lo que se dio en clase, colocando referencias claras.
El orden de los ejercicios es arbitrario. Recomendamos leer el parcial completo antes de empezar a resolverlo.

Ejercicio 1 - Programación funcional

Aclaración: en este ejercicio no está permitido utilizar recursión explícita, a menos que se indique lo contrario.

El siguiente tipo de datos sirve para representar árboles ternarios:

```
data AT a = NilT | Tri a (AT a) (AT a) (AT a)
```

Definimos el siguiente árbol para los ejemplos:

```
at1 = Tri 1 (Tri 2 NilT NilT NilT) (Tri 3 (Tri 4 NilT NilT NilT) NilT NilT)
      (Tri 5 NilT NilT NilT)
```

a) Dar el tipo y definir la función `foldAT` que implementa el esquema de recursión estructural para el tipo `AT a`. Sólo en este inciso se permite usar recursión explícita.

b) Definir la función `preorder :: AT a -> [a]`, que lista los nodos de un árbol ternario en el orden en que aparecen: primero la raíz, después los nodos del subárbol izquierdo, luego los del medio y finalmente los del derecho.

   Por ejemplo: `preorder at1` ⇝ `[1, 2, 3, 4, 5]`.

c) Definir la función `mapAT :: (a -> b) -> AT a -> AT b`, análoga a la función map para listas, pero para árboles ternarios.

   Por ejemplo: `mapAT (+1) at1` ⇝ `Tri 2 (Tri 3 NilT NilT NilT) (Tri 4 (Tri 5 NilT NilT NilT) NilT NilT) (Tri 6 NilT NilT NilT)`.

d) Definir la función `nivel :: AT a -> Int -> [a]`, que devuelve la lista de nodos del nivel correspondiente del árbol, siendo 0 el nivel de la raíz.

   Por ejemplo: `nivel at1 1` ⇝ `[2, 3, 5]`.

   Pista: aprovechar la currificación y utilizar evaluación parcial.

Ejercicio 2 - Demostración de propiedades

Considerar las siguientes definiciones sobre listas y árboles estrictamente binarios[1]:

```
data AEB a = Hoja a | Bin (AEB a) a (AEB a)

    const :: a -> b -> a
{C} const = (\ x -> \ y -> x)

    head :: [a] -> a
{H} head (x:xs) = x

    tail :: [a] -> [a]
{T} tail (x:xs) = xs

     length :: [a] -> Int
{L0} length [] = 0
{L1} length (x:xs) = 1 + length xs

     null :: [a] -> Bool
{N0} null [] = True
{N1} null (x:xs) = False
```

[1] Escritas con recursión explícita para facilitar las demostraciones.

```
     altura :: AEB a -> Int
{A0} altura (Hoja x) = 1
{A1} altura (Bin i r d) = 1 + max (altura i) (altura d)

     esPreRama :: Eq a => AEB a -> [a] -> Bool
{E0} esPreRama (Hoja x) = \xs -> null xs || (xs == [x])
{E1} esPreRama (Bin i r d) = \xs -> null xs ||
                       (r == head xs && (esPreRama i (tail xs) || esPreRama d (tail xs)))
```

a) Asumiendo `Eq a`, demostrar la siguiente propiedad:

$\forall$ `t::AEB a` . $\forall$ `xs::[a]` . esPreRama t xs $\Rightarrow$ length xs $\leq$ altura t

Se recomienda hacer inducción en el árbol, utilizando extensionalidad de booleanos y listas cuando sea necesario. Se permite definir macros (i.e., poner nombres a expresiones largas para no tener que repetirlas).

No es obligatorio escribir los $\forall$ correspondientes en cada paso, pero es importante recordar que están presentes. Recordar también que los $=$ de las definiciones pueden leerse en ambos sentidos.

Se consideran demostradas todas las propiedades conocidas sobre enteros y booleanos, así como también que $\forall$ `t::AEB a` . altura t ≥ 0.

b) Demostrar el siguiente teorema usando deducción natural, sin utilizar principios clásicos:

$((P \Rightarrow Q) \wedge (Q \Rightarrow R)) \Rightarrow \neg R \Rightarrow \neg P$

Ejercicio 3 - Cálculo Lambda Tipado

Se desea extender el cálculo lambda simplemente tipado para modelar **Diccionarios**. Para eso se extienden los tipos y expresiones de la siguiente manera:

$\tau ::= \cdots \mid \texttt{Dicc}(\tau, \tau)$

$M ::= \cdots \mid \texttt{Vacío}_{\sigma,\tau} \mid \texttt{definir}(M, M, M) \mid \texttt{def?}(M, M) \mid \texttt{obtener}(M, M)$

- $\texttt{Dicc}(\sigma, \tau)$ es el tipo de los diccionarios con claves de tipo σ y valores de tipo τ.
- $\texttt{Vacío}_{\sigma,\tau}$ es un diccionario vacío con claves de tipo σ y valores de tipo τ.
- $\texttt{definir}(M, N, O)$ define el valor O en el diccionario M para la clave N.
- $\texttt{def?}(M, N)$ indica si la clave N fue definida en el diccionario M.
- $\texttt{obtener}(M, N)$ da el valor asociado a la clave N en el diccionario M (se espera que el diccionario tenga definida la clave y, en caso contrario, la expresión puede tipar, pero no se obtendrá un valor).

a. Introducir las reglas de tipado para la extensión propuesta.

b. Definir el conjunto de valores y las nuevas reglas de semántica operacional a pequeños pasos. Suponer que el tipo de las claves cuenta con el operador == (es decir, el cálculo está extendido con un operador de comparación para los tipos que se usen como claves). Para el caso de $\texttt{obtener}(M, N)$, se espera que la clave N esté definida en el diccionario M. Si no lo está, puede colgarse o terminar en una expresión de error (una forma normal que no es un valor). **No es necesario escribir las reglas de congruencia**, sino que basta con indicar cuántas son.

c. Mostrar paso por paso cómo reduce la expresión:

$(\lambda d\colon \texttt{Dicc}(\mathsf{Nat}, \mathsf{Bool}).\text{if } \texttt{def?}(d, \underline{0}) \text{ then } \texttt{obtener}(d, \underline{0}) \text{ else False})\ \texttt{definir}(\texttt{Vacío}_{\mathsf{Nat},\mathsf{Bool}}, \underline{0}, \mathsf{True})$

Suponer que zero $==$ zero $\to$ True.

Corrigió Daniela

OTAZUA ARCE, MATEO (88/23) (0:185)

HOJA 1/4

Ej	1	2	3
	B	R	B-

Aprobado

Ejercicio 1

a)
```
foldAT :: b -> (a -> b -> b -> b -> b) -> AT a -> b
foldAT cNil cTri t = case t of
        NilT -> cNil
    Tri x i m d -> cTri x (rec i) (rec m) (rec d)
        where rec = foldAT cNil cTri
```
✓

b)
```
preorder :: AT a -> [a]
preorder = foldAT [] (\x ri rm rd -> x:(ri++rm++rd))
```
✓

c)
```
mapAT :: (a->b) -> AT a -> AT b
mapAT f = foldAT (NilT) (\x ri rm rd -> Tri (f x) ri rm rd)
```
✓

d)
```
nivel :: AT a -> Int -> [a]
nivel = foldAT (\i -> []) (\x ri rm rd -> \i ->
                        if i==0
                           then [x]
                           else ((ri i-1)++(rm i-1)++(rd i-1)))
```
= const []

✓

(muy prolijo ü)

Ejercicio 2

a) Se nos pide demostrar la siguiente propiedad:

$\forall t : AEB\ a ::$

$\quad \forall xs : [a] ::$

$\quad\quad$ esPrerama t xs $\rightarrow$ length xs $\leq$ altura t

Para demostrarla, hacemos inducción estructural sobre el árbol (AEB). Tiene dos constructores: Hoja y Bin.

Empezamos por el CASO BASE (Hoja h). $\forall h : a ::$ P(Hoja h).

Defino P(x) = $\forall xs : [a] ::$
esPrerama x xs $\rightarrow$ length xs $\leq$ altura x.

P(Hoja h) = $\forall xs : [a] ::$
esPrerama (Hoja h) xs $\rightarrow$ length xs $\leq$ altura (Hoja h)

Reemplazo por {A0} y {E0}.

$\forall xs : [a] ::$
(\xs -> null xs || (xs == [h])) xs -> length xs $\leq$ 1

Aplico β

$\forall xs : [a] ::$
null xs || (xs == [h]) -> length xs $\leq$ 1

No hacía falta, se podía usar extensionalidad (lo aclara la consigna)

Para demostrar esto hacemos inducción sobre listas. Definimos la anterior expresión sin el $\forall$ como Q(xs).

CASO BASE Q([])

null [] || ([] == [h]) -> length [] $\leq$ 1

Reemplazamos por {N0} y {L0}

True || ([] == [h]) -> 0 $\leq$ 1

Simplificamos. {||} y {$\leq$}

True -> True ✓

Queda demostrado Q([]).

PASO INDUCTIVO Q(ys) -> Q(y:ys)

Asumimos Q(ys). Demostramos Q(y:ys).

null (y:ys) || ((y:ys) == [h]) -> length (y:ys) $\leq$ 1

Reemplazamos por {N1} y {L1}.

False || ((y:ys) == [h]) -> 1 + length (ys) $\leq$ 1

Simplificamos. {||}

(y:ys) == [h] -> 1 + length (ys) $\leq$ 1

Ahora podemos separar en dos casos: ys es lista vacía o ys tiene al menos un elemento.

Si ys es vacía, reemplazando por {L0} en la derecha, nos queda la tautología 1 $\leq$ 1, ~~así que~~ así que no importaría lo que quede a la izquierda de la implicación.

Si ys tiene al menos un elemento, entonces (y:ys) == [h] nunca podría ser verdadero, así que no importaría lo que quede del lado derecho.

Queda demostrado Q(y:ys) [illegible] Q(ys)

Habiendo demostrado $Q([\,])$ y $Q(y:ys)$, queda demostrado $\forall h : a :: P(\text{Hoja } h)$.

Ahora avanzamos con el PASO INDUCTIVO (Bin i r d).

$$(\forall i, d : AEB\,a :: P(i) \land P(d)) \rightarrow (\forall r : a :: P(\text{Bin } i\ r\ d))$$

Asumimos verdadero la izquierda de la implicación.

$P(\text{Bin } i\ r\ d) = \forall xs : [a] ::$
esPrecama (Bin i r d) xs $\rightarrow$ length xs $\leq$ altura (Bin i r d)

Veamos por separado los lados de la implicación. Primero la izquierda.

esPrecama (Bin i r d) xs
{E1 + β}
null xs || (r == head xs && (esPrecama i (tail xs) || esPrecama d (tail xs))

Ahora la derecha.

length xs $\leq$ altura (Bin i r d)
{A1}
length xs $\leq$ ~~altura~~ 1 + max (altura i) (altura d)

Juntando ambas partes queda

$\forall xs : [a] ::$
null xs || (r == head xs && (esPrecama i (tail xs) || esPrecama d (tail xs))) $\rightarrow$ length xs $\leq$ 1 + max (altura i) (altura d)

Para demostrar esto aplicamos extensionalidad sobre listas:

CASO BASE xs = []

null [] || (r == head [] && (esPrecama i (tail []) || esPrecama d (tail []))) $\rightarrow$ length [] $\leq$ 1 + max (altura i) (altura d)
{N0}, {L0} y {I1}

True $\rightarrow$ 0 $\leq$ 1 ~~+~~ max (altura i) (altura d).

Como $\forall t : AEB :: \text{altura } t \geq 0$, lo ~~de~~ la derecha es una tautología y vale ~~entonces~~ la implicación. V

CASO xs = y:ys

null (y:ys) || (r == head (y:ys) && (esPrecama i (tail (y:ys)) || esPrecama d (tail (y:ys))) $\rightarrow$ length (y:ys) $\leq$ 1 + max (altura i) (altura d)
{N1}, {H} {T1} {L1}

(r == head && (esPrecama i ys || esPrecama d ys) $\rightarrow$ 1 + length ys $\leq$ 1 + max (altura i) (altura d)

Extensionalidad de booleanos? ←

Ahora tenemos dos casos: $r = y$, $r \neq y$
Si $r \neq y$, lo de la izquierda es siempre falso, y vale la implicación.
Si $r = y$
etc ??

Incompleto.

Ejercicio 2

b)

Incompleto.

$$\Gamma$$

$$\Gamma, P \vdash Q \qquad \Gamma, P \vdash Q \to R \quad \wedge e_2 \text{ etc}$$

$$\Gamma, P \vdash R$$

$$\Gamma \vdash P \to R \quad \to \qquad \Gamma \vdash \neg R \quad ax$$

$$\Gamma = (P \to Q) \wedge (Q \to R), \neg R \vdash \neg P \quad MT$$

$$(P \to Q) \wedge (Q \to R) \vdash \neg R \to \neg P \quad \to_i$$

$$\vdash ((P \to Q) \wedge (Q \to R)) \to \neg R \to \neg P \quad \to_i$$

Ejercicio 3

$\tau ::= \ldots \mid Dicc(\tau, \tau)$

$M ::= \ldots \mid Vacío_{\sigma,\tau} \mid definir(M,M,M) \mid def?(M,M) \mid obtener(M,M)$

a) REGLAS DE TIPADO

$$\frac{}{\Gamma \vdash Vacío_{\sigma,\tau} : Dicc(\sigma,\tau)}\, ax_{vacío}$$

$$\frac{\Gamma \vdash M : Dicc(\sigma,\tau) \quad \Gamma \vdash N:\sigma \quad \Gamma \vdash O:\tau}{\Gamma \vdash definir(M,N,O) : Dicc(\sigma,\tau)}\, definir$$

$$\frac{\Gamma \vdash M : Dicc(\sigma,\tau) \quad \Gamma \vdash N:\sigma}{\Gamma \vdash def?(M,N) : Bool}\, def?$$

$$\frac{\Gamma \vdash M : Dicc(\sigma,\tau) \quad \Gamma \vdash N:\sigma}{\Gamma \vdash obtener(M,N) : \tau}\, obtener$$

b)

$V ::= \ldots \mid Vacío_{\sigma,\tau} \mid definir(V,V,V)$

SEMÁNTICA OPERACIONAL

(def?V) $def?(Vacío_{\sigma,\tau}, W) \to False$

(def?X) $def?(definir(V,W,U), W') \to$ if $W == W'$ then True else $def?(V,W')$

(obtX) $obtener(definir(V,W,U), W') \to$ if $W == W'$ then U else $obtener(V,W')$

(obtV) $obtener(Vacío_{\sigma,\tau}, W) \to obtener(Vacío_{\sigma,\tau}, W)$ // (se cuelga)

y hay 7 reglas de congruencia: 3 para definir(), 2 para def? y 2 para obtener()

c) $\longrightarrow$

~~Redacción~~

C) Reducimos la siguiente expresión:

$(\lambda d: Dicc(Nat, Bool).\ \text{if } def?(d, \underline{0}) \text{ then } obtener(d, \underline{0}) \text{ else } False)\ definir(Vacío_{Nat,Bool}, \underline{0}, True)$

$\xrightarrow{(\beta_1)}$ $(\text{if } def?(d, \underline{0}) \text{ then } obtener(d, \underline{0}) \text{ else } False)\{d := definir(Vacío_{Nat,Bool}, \underline{0}, True)\}$

$\cancel{(\beta_2)} =$ $\text{if } def?(definir(Vacío_{Nat,Bool}, \underline{0}, True), \underline{0}) \text{ then } obtener(definir(Vacío_{Nat,Bool}, \underline{0}, True), \underline{0}) \text{ else } False$

$\xrightarrow{(if_c + defX)}$ $\text{if } (\text{if } \underline{0} == \underline{0} \text{ then } True \text{ else } def?(Vacío_{Nat,Bool}, \underline{0})) \text{ then } obtener(definir(Vacío_{Nat,Bool}, \underline{0}, True), \underline{0}) \text{ else } False$

$(if_c + if_c + \underline{0} = \underline{0} + if_t)$ SALTEA PASOS ⊛

$\longrightarrow$ $\text{if } True \text{ then } obtener(definir(Vacío_{Nat,Bool}, \underline{0}, True), \underline{0}) \text{ else } False$

$\xrightarrow{(if_t)}$ $obtener(definir(Vacío_{Nat,Bool}, \underline{0}, True), \underline{0})$

$\xrightarrow{(obtX)}$ $\text{if } \underline{0} == \underline{0} \text{ then } True \text{ else } obtener(Vacío_{Nat,Bool}, \underline{0})$

$\xrightarrow{(\underline{0} = \underline{0} + if_t)}$ True

γ

⊛ debería ser

→ if (if true then ...) then ...